M&G No. 50037.218US01

METHOD AND SYSTEM FOR DATA BINDING IN A BLOCK STRUCTURED USER INTERFACE SCRIPTING LANGUAGE

Background of the Invention

A User Interface (UI) scripting language defines the layout of UI components on the output of a computing device. Typical components of the UI might be menus, buttons or check boxes. For each component there is usually additional information such as position, size, color and textual or pictorial content.

Typically, as with other types of scripting languages, a UI written in a block structured UI scripting language is often hand-authored to include the desired components for display and the layout of those components. However, hand-authoring the UI limits the UI to the components and the layout defined at the time of authoring creating difficulties in adding certain dynamic data to the UI. For example, a set of links to web pages may be included on the UI that are dynamic and generated from an external source (e.g., a server) at run time. Run time is defined as the time during which the User Interface component is being displayed or used. Custom code could be written and then spliced into the UI for displaying the dynamic data, but this method may be cumbersome and difficult in a block structured scripting language. What is needed is a way to update data from an external data source in a block structured UI scripting language without requiring the use of custom code.

Summary of the Invention

The present invention provides a method and system for an extension of a User Interface scripting language in order to allow the attributes of any component to be repeatedly updated at "run time" from an external data source. An example of an external data source might be a database on a separate computer.

In one embodiment, the present invention includes three tags that are used within XML (Extensible Markup Language). The three tags may be referred to as

the "Reference" tag, the "ReferenceTemplate" tag, and the "ReferenceMerge" tag, however other names may be used. The Reference tag indicates UI script that has associated data binding elements. The ReferenceTemplate tag and the ReferenceMerge tag implement the data binding functionality.

In a further embodiment, the UI script is transformed to a tree structure. The relevant portions of the tree structure are cloned and manipulated to insert the data from the external source. The cloned portions of the tree structure are then grafted back into the tree. The tree structure is then used to display the UI corresponding to the UI script that includes the data from the external source. In one embodiment, the cloned portions of the tree correspond to the portions of the tree associated with Reference-Template and ReferenceMerge tags.

Brief Description of the Drawings

FIGURE 1 illustrates an exemplary computing device that may be used in one exemplary embodiment of the present invention.

FIGURE 2 illustrates an exemplary block diagram for a data transformation pipeline in accordance with the present invention.

FIGURE 3 illustrates an exemplary state table and scenarios for presenting UI depending on the server state in accordance with the present invention.

FIGURE 4 illustrates an exemplary set of data, exemplary XML for UI display of the data, and the resultant output of the XML in accordance with the present invention.

FIGURE 5 is a flow diagram for an exemplary process for binding data to in a UI script in accordance with the present invention.

Detailed Description

The present invention now will be described more fully hereinafter with reference to the accompanying drawings, which form a part hereof, and which show, by way of illustration, specific exemplary embodiments for practicing the invention. This invention may, however, be embodied in many different forms and should not be

construed as limited to the embodiments set forth herein; rather, these embodiments are provided so that this disclosure will be thorough and complete, and will fully convey the scope of the invention to those skilled in the art. Among other things, the present invention may be embodied as methods or devices. Accordingly, the present invention may take the form of an entirely hardware embodiment, an entirely software embodiment or an embodiment combining software and hardware aspects. The following detailed description is, therefore, not to be taken in a limiting sense.

Illustrative Operating Environment

FIGURE 1 shows an exemplary computing device that may be included in system **100** for implementing the invention. Computing device **100** illustrates a general operating environment that may apply to the present invention. In a very basic configuration, computing device **100** typically includes at least one processing unit **102** and system memory **104**. Processing unit **102** includes existing physical processors, those in design, multiple processors acting together, virtual processors, and any other device or software program capable of interpreting binary executable instructions. Depending on the exact configuration and type of computing device, the system memory **104** may be volatile (such as RAM), non-volatile (such as ROM, flash memory, etc.) or some combination of the two. System memory **104** typically includes an operating system **105**, one or more program modules **106**, and may include program data **107**. This basic configuration is illustrated in FIGURE 1 by those components within dashed line **108**.

Computing device **100** may also have additional features or functionality. For example, computing device **100** may also include additional data storage devices (removable and/or non-removable) such as, for example, magnetic disks, optical disks, or tape. Such additional storage is illustrated in FIGURE 1 by removable storage **109** and non-removable storage **110**. Computer storage media may include volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information, such as computer readable instructions, data structures, program modules or other data. System memory **104**,

removable storage **109** and non-removable storage **110** are all examples of computer storage media. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by computing device **100**. Any such computer storage media may be part of computing device **100**. Computing device **100** may also have input device(s) **112** such as keyboard, mouse, pen, stylus, voice input device, touch input device, etc. Output device(s) **114** such as a display, speakers, printer, etc. may also be included. All these devices are known in the art and need not be discussed at length here.

Computing device **100** may also contain communications connection(s) **116** that allow the device to communicate with other computing devices **118**, such as over a network. Communications connection(s) **116** is an example of communication media. Communication media typically embodies computer readable instructions, data structures, program modules or other data in a modulated data signal such as a carrier wave or other transport mechanism and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared and other wireless media. The term computer readable media as used herein includes both storage media and communication media.

Data Binding

The present invention is generally directed to retrieving data from an external source and binding the data to a structure used in executing a UI script. The external source may be on a local machine located across a network. The retrieval of the data from the external source may happen once (e.g., when the rest of the UI is loaded) or many times over the lifetime of the UI.

FIGURE 2 illustrates an exemplary block diagram for a data transformation pipeline in accordance with the present invention. Block diagram **200** includes data **202**, optional server-side transform **204**, communication library **206**, optional client-side transform **208**, client-side template **210**, and UI outputs **212**. Data **202** and optional server-side transform **204** are located on server **220**.

Communication library **206** may be used to retrieve data **202** from the remote location. Depending on the type of data, certain security precautions may be used in transmitting the data. In one embodiment, secure communication is required for any personal data, and publicly available data is more easily transmitted in clear text.

To retrieve data **202**, a URL (uniform resource locator) is passed from client-side template **210** to communication library **206**. Communication library **206** then requests data **202** according to the URL from server **220**, which may have gone through a server transformation through optional server-side transform **204**. If data **202** is transformed on the server side, client-side transform **208** retransforms data **202** before sending the data through client-side template **210** to produce UI outputs **212**. In another embodiment, the script may also be produced server-side instead of using client-side template **210**. By not specifying a template on the client, the raw script from server **220** is inserted directly into a document tree used for producing the UI output.

FIGURE 3 illustrates an exemplary state table and scenarios for presenting UI depending on the server state in accordance with the present invention. State table **300** includes the following states: default **302**, online **304**, offline **306**, empty **308**, and error **308**. Additionally, a refresh of the UI is indicated by refresh **320**, wherein the UI is refreshed to update data and other changes.

Each data binding tag is in one state at a time, and moves from state to state as the connection or server situation changes. The present invention provides a wide range of flexibility by allowing users to supply different templates (and even different data) per state.

Default **302** is the state denoted by the empty string. Default **302** is the state any data binding is considered to be in until another state is achieved. In particular, when data is remote, it may take a few seconds to reach the "online" state,

during which time the default state is shown. Default **302** includes content indicating that the actual content is in the process of being retrieved.

Online **304** indicates the presence of data that refreshes as often as specified by an interval attribute that may be included in a tag of the UI script. Offline **306** indicates that the data source defined in online **304** can't be retrieved because the server can't be contacted (e.g., because of a failure in connecting to the network, as different from a failure at the server). Offline **306** allows a secondary data source (e.g. an offline cache, etc.) to be defined to attempt a secondary retrieval of the data. Empty **308** indicates that the server returned a response that no data matched the query. Reaching the empty state means that records were expected but not found.
Error **310** is reached when a problem with the server prevented the data from being retrieved and an alternative action is not triggered. Error **310** may be reached due to a timeout or corrupted return value.

Whenever a state transition occurs, a refresh timer of refresh **320** is reset to the value of the interval attribute of the new state. If the new state does not specify an interval, then the interval attribute of the online state is used. In one implementation of data binding the refresh timer is not reset if the interval value is zero, negative or not provided in the script.

As previously indicated, one embodiment of the present invention operates within the context of an extensible markup language (XML). A scripting language that conforms to the rules of XML consists of a set of tags. Each tag may have a set of zero or more attributes that modify it. In a User Interface scripting language the tags typically refer to types of user interface components such as menus, buttons, check boxes, edit boxes, text boxes, lists or links. The attributes of a tag typically convey such information as position, size, color and textual or pictorial content of the component associated with the tag. A script is a document (or set of documents) written in the scripting language. It defines the form, function and position of a set of components. For example, a script consisting of two components, a label with the text "Michelangelo" and a checkbox for "allow email", might appear as follows:

<label fontsize="8" font="Times New Roman" text="Michaelangelo" />
<checkbox state="checked" text="Allow email" />

A tag may also have a scope that may or may not include the scopes of other tags. Thus the scope of a tag may precede or follow the scope of another tag, or it may be wholly contained or "nested" within the scope of another tag. The scope of a tag exists from the occurrence of the tag in the script to the occurrence of an "end tag" associated with the given tag. This is similar in concept to the behavior of brackets or parentheses in natural language. In XML, end tags have the same name as their corresponding tag but are preceded by a slash character. Thus </text> is the end tag for <text>.

The purpose of nested scopes is to allow components to be elements of other components. For example, a list of three checkboxes might appear as follows. Note that the scope of the list tag includes the three checkbox tags.

```
<list>
    <checkbox text="Apples" />
    <checkbox text="Oranges" />
    <checkbox text="Bananas" />
</list>
```

In one embodiment, the present invention includes three tags that are used within the XML previously described. The three tags may be referred to as the "Reference" tag, the "ReferenceTemplate" tag, and the "ReferenceMerge" tag. Both the ReferenceTemplate tag and the ReferenceMerge tag have attributes. In one embodiment, these three tags do not themselves refer to UI components, but rather to the application of the data binding procedure to the tags that are within their scope. The ReferenceMerge tag appears inside the scope of a ReferenceTemplate tag and, similarly, the ReferenceTemplate tag appears inside the scope of a Reference tag. The

set of tags within the scope of either a ReferenceTemplate tag or a ReferenceMerge tag is known as its "template".

FIGURE 4 illustrates an exemplary set of data, exemplary XML for UI display of the data, and the resultant output of the XML in accordance with the present invention. The exemplary data **410** includes two articles (**412, 414**) that are identified by a title (e.g., **416**) that is a hyperlink to the article according to the specified URL (e.g., **418**). The exemplary XML **420** for display of the hyperlink data includes a Reference tag (e.g., **422**), a ReferenceTemplate tag (e.g., **424**), and a ReferenceMerge tag (e.g., **426**). The resultant output **450** produces two hyperlinks with titles "Mail Merge without the Misery" and "How to Succeed in Tools | Options without Really Trying".

As may be seen from the example, the Reference tag (e.g., **422**) has no attributes. The scope of a Reference tag is used to delineate those components to which the data binding is applied. In addition, each Reference tag includes an internal state, or "state of the reference", which may have any of several values. One instance of data binding defines the values "default", "online", "offline", "empty" and "error" as described in grater detail in the discussion of FIGURE 3 above.

For any given state, the Reference tag (e.g., **422**) may contain within its scope either zero or one occurrences of a ReferenceTemplate tag (e.g., **424**) associated with that state. The scope of the ReferenceTemplate tag delineates those components that are included in the layout when that state is in effect. If the current state of the reference does not match the state associated with a given ReferenceTemplate tag then the components within the scope of the ReferenceTemplate tag are not displayed. The state of a ReferenceTemplate tag is determined by its "state" attribute (e.g., **428**).

Another important attribute of the ReferenceTemplate tag is the "source" attribute (e.g., **430**). The value of this attribute is the URL (Universal Resource Locator) string that specifies the data source that is queried for data.

Also shown in the XML **420** for displaying the hyperlink data are the placeholders (e.g., **432**) indicating where the data is to be included. A placeholder is a reference to data returned from the external data source. When the data is obtained, the

data is substituted for the placeholder before the component is created. In one implementation, it is assumed that data from an external data source is returned in XML format. In this implementation a placeholder has one of the following forms,

```
@tag@
@tag;attribute@
```

where tag and attribute are the names of actual tags or attributes in the XML data. Note that the placeholder is identified by being surrounded by a "placeholder character" and if an attribute is specified it is separated from the tag by a "separator character". In the above example the placeholder character is the "@" sign and the separator character is the semicolon.

In another embodiment, the ReferenceTemplate tag (e.g., **424**) may have an optional "interval" attribute that specifies how often the data binding is to be performed. The interval attribute is useful when the data in the external data source is changing with time. For example, the following ReferenceTemplate element would be updated every five seconds.

```
<ReferenceTemplate state="online" interval="5">
    <label text="@name@" />
</ReferenceTemplate>
```

When a single data record is expected from a data source the ReferenceTemplate tag is sufficient to define a set of components that depend on the data.

However, as in the example shown, it is sometimes the case that a set of records rather than a single record can be retrieved from a data source, where the number of records returned cannot be predicted ahead of time. For instance, one might query an employee database for the set of records of employees that match some criterion. The ReferenceMerge tag (e.g., **426**) handles multiple records. The scope of a

ReferenceMerge tag defines a set of components that will be replicated and instantiated with data for each individual record that is obtained from the data source.

To control excessive replication when a large number of records is returned, it is possible to specify a value for a "maxrecords" attribute (e.g., **434**) of the ReferenceMerge tag. In the present example, only five hyperlinks would be included in the list for display on the UI despite the number included in data **410**.

In another embodiment, the ReferenceMerge tag may also have a "SkipPartialRecords" tag which has the value true or false. If set to true then partial records are skipped. If SkipPartialRecords is set to false (or not given) the record would be partially displayed in the UI despite missing a portion of the record.

The user interface script is usually transformed into a visible display by first converting it to an internal data structure known as a tree. The script and its corresponding tree are logically equivalent. Each form can be converted to the other without loss of essential information. The purpose of the tree structure is to make it easier to modify the elements of the display. It is computationally simpler and faster to add or delete elements of a tree than it is to edit a script.

The tags in the script become nodes of the tree and the attributes of the tags become properties of the nodes. In addition the scope relationships among the tags are modeled by the tree structure. Sequential tags become siblings in the tree and tags directly within the scope of an enclosing tag become child nodes of the enclosing tag's node. Ultimately the contents of the tree are converted to visible elements on the screen.

Generally when a script is processed the tags are immediately converted to nodes in the internal tree structure. However this behavior is not appropriate for tags within a data binding template since they may contain placeholders rather than actual data. A typical implementation is to store the template as a property of the data binding node rather than as a sub-tree (or set of sub-trees) of the node. It is possible to store the template in its original script format but it is generally more useful to convert it to its equivalent tree structure so that it becomes a disjoint "branch" that can be grafted to the main tree whenever it is convenient.

Once parsing of the script has been completed but before the tree is displayed, the templates of any default ReferenceTemplate nodes are retrieved and grafted to the tree. Then the tree can be displayed for the first time. Following this, the source attribute of the online ReferenceTemplate node of each reference is retrieved and a query is made to access the external data source. A link is made for each query to the reference from which it originated.

When a response from a query is received the form of the response will determine the state to which the reference should change. If no ReferenceTemplate template exists for the indicated state then no further action is taken and the state is not changed. If a template exists then either it is displayed immediately if no data binding is required, or else the data binding process is begun with the given template.

FIGURE 5 is a flow diagram for an exemplary process for binding data to in a UI script in accordance with the present invention. Process **500** begins at a start block wherein a UI script has been written such that data binding from an external data source occurs prior to display of the UI. Processing continues at block **502**.

At block **502**, any previous clone templates that were grafted to the tree are removed from the tree. Removing the previously grafted clone templates from the tree allows the data binding process to commence with a tree where data binding has not yet occurred to avoid errors in the process. Once these previously cloned templates are removed, processing continues at block **504**.

At block **504**, the relevant ReferenceTemplate template is copied or "cloned". Cloning the template allows the original template to be available for subsequent iterations of the data binding process. After the template is cloned, processing continues at block **506**.

At block **506**, the placeholders within the UI script are located and associate with the corresponding data from the external data source. The process for retrieving the data from the data source is described in greater detail with relation to FIGURE 2 above. Processing proceeds to block **508**.

At block **508**, the placeholders within the UI script are replaced with the data from the external data source in the cloned copy of the template. The cloned copy

of the template now reflects the UI script with the data inserted. Processing continues at block **510**.

At block **510** the cloned template is grafted to the tree of the UI script. In one embodiment, the cloned template replaces any previously grafted template. Processing then continues at decision block **512**.

At decision block **512**, a determination is made whether the ReferenceTemplate template includes a ReferenceMerge template or if the ReferenceMerge template has already been addressed. If the ReferenceTemplate template includes a ReferenceMerge template, processing moves to decision block **514**. However, if the ReferenceTemplate template does not include a ReferenceMerge template, processing advances to decision block **516**.

At decision block **514**, a determination is made whether any MaxRecords attribute associated with the ReferenceMerge template has been met. For example, the MaxRecords attribute may indicate that only five records are to be retrieved from the external data source. Accordingly, when the MaxRecords attribute, no more records are retrieved. If the MaxRecords attribute associated with the ReferenceMerge template has not been met, processing returns to block **504** where the process is repeated for the next data record. However, if the MaxRecords attribute associated with the ReferenceMerge template has been met, processing advances to decision block **516**.

At decision block **516**, a determination is made whether an interval is associated with the template. If no interval is associated, processing continues to end block where process **500** ends. However, if an interval does exist, processing moves to decision block **518**.

At decision block **518**, a determination is made whether a timer associated with the interval has expired. The relevant interval value is retrieved, usually from either the interval attribute of the current state or the online state, and then a timer is started using the given value. When the timer goes off, processing returns to block **502** where process **500** is repeated. Otherwise, processing holds at decision block **518** where the state of the timer is monitored.

The above specification, examples and data provide a complete description of the manufacture and use of the composition of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.